Introduction to the CLASP Process

Dan Graham, Secure Software, Inc.

2006-11-16 L3 / L, M[1]

Comprehensive, Lightweight Application Security Process—CLASP—is an activity-driven, role-based set of process components guided by formalized best practices. CLASP is designed to help software development teams build security into the early stages of existing and new-start software development life cycles in a structured, repeatable, and measurable way.

CLASP is based on extensive field work by Secure Software employees in which the system resources of many development life cycles were decomposed to create a comprehensive set of security requirements. These resulting requirements form the basis of CLASP's Best Practices, which can enable organizations to systematically address vulnerabilities that, if exploited, can result in the failure of basic security services (e.g., confidentiality, authentication, and authorization).

Overview

This introduction to the CLASP Process provides an overview of CLASP's process structure and the dependencies among the CLASP process components, as follows:

- CLASP Views
- CLASP Best Practices
- 24 CLASP Activities
- CLASP Resources (including CLASP Keywords)
- Taxonomy of CLASP

This introduction concludes with a section providing further information on CLASP.

CLASP Views

The CLASP Process is presented through five high-level perspectives called *CLASP Views*. These views allow CLASP users to quickly understand the CLASP Process, including how CLASP process components interact and how to apply them to a specific software development life cycle.

These are the CLASP Views:

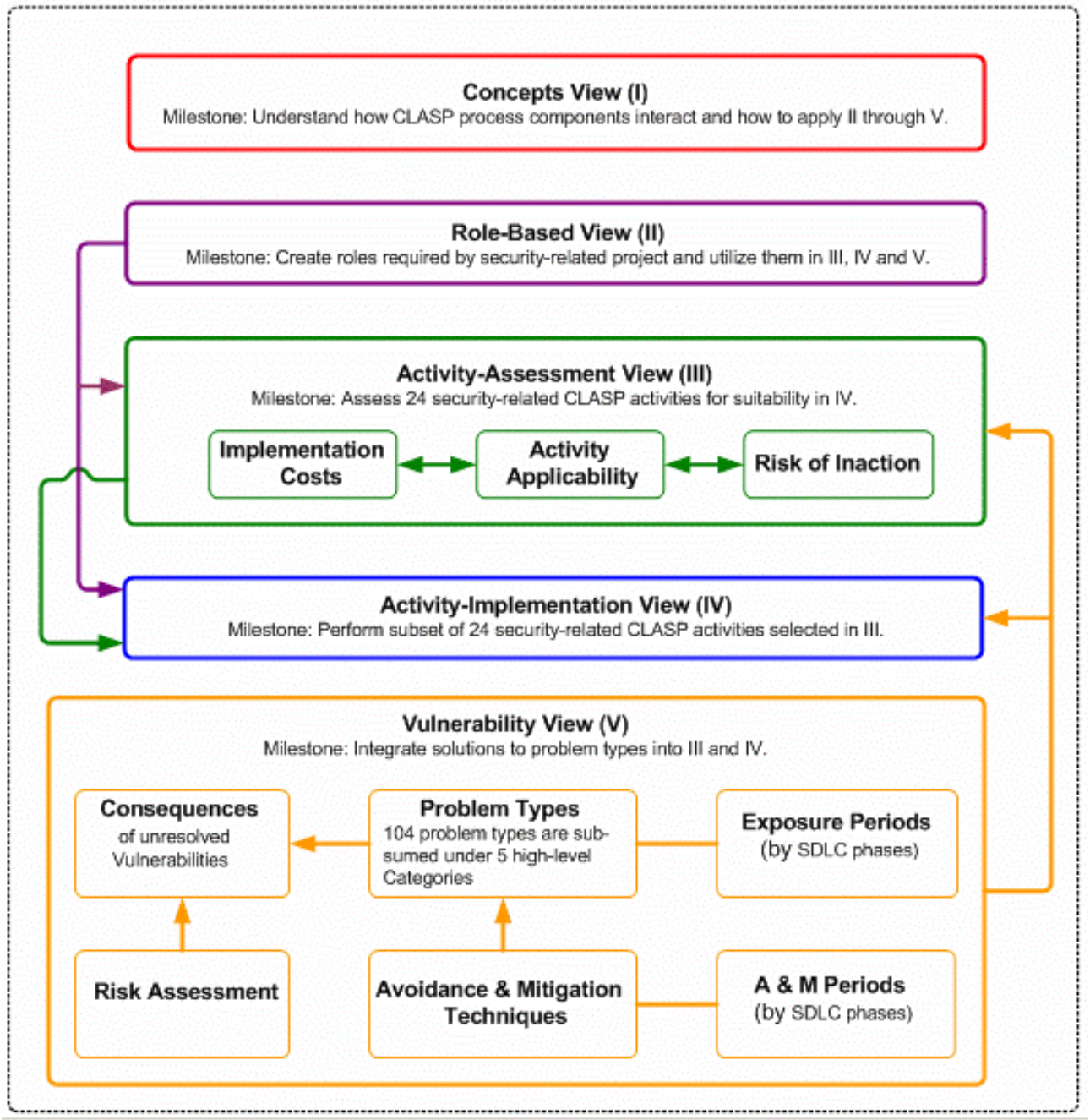
- **Concepts View**
  This view provides a high-level introduction to CLASP by briefly describing, for example, the interaction of the five CLASP Views, the seven CLASP Best Practices, the CLASP Taxonomy, the relation of CLASP to security policies, and a sample sequence for applying CLASP process components.
- **Role-Based View**
  This view contains role-based introductions to the CLASP Process.
- **Activity-Assessment View**
  This view helps project managers assess the appropriateness of the 24 CLASP Activities and select a subset of them. CLASP provides two sample road maps (i.e., legacy and new-start) to help select applicable activities.
- **Activity-Implementation View**
  This view contains the 24 security-related CLASP Activities that can be integrated into a software development process. The activities phase of the SDLC translates into executable software any subset of the 24 security-related activities assessed and accepted in Activity Assessment.
- **Vulnerability View**

This view contains a catalog of the 104 underlying “problem types” identified by CLASP that form the basis of security vulnerabilities in application source code. CLASP divides the 104 problem types into 5 high-level categories. An individual problem type in itself is often not a security vulnerability; frequently, it is a combination of problems that create a security condition leading to a vulnerability in source code.

Associated with the Vulnerability View are the CLASP Vulnerability Use Cases, which depict conditions under which security services are vulnerable to attack at the application layer. The use cases provide CLASP users with easy-to-understand, specific examples of the relationship between security-unaware source coding and possible resulting vulnerabilities in basic security services.

Figure 1 shows the CLASP Views and their interactions.

Figure 1. CLASP views and their interactions



CLASP Best Practices

Within a software development project, the CLASP Best Practices are the basis of all security-related software development activities—whether planning, designing or implementing—including the use of all tools and techniques that support CLASP.

These are the seven CLASP Best Practices:

1. **Institute awareness programs**
   Essential security concepts and techniques may be foreign to an organization's software developers and others involved in application development and deployment. Thus, it is imperative at the outset to educate everyone involved. It is critical that project managers—as the driving force behind most application development or upgrade projects—consider security to be an important project goal, both through training and accountability. Awareness programs can be readily implemented, using external expert resources as appropriate, and can deliver a high return by helping to ensure that other activities promoting secure software will be implemented effectively.
2. **Perform application assessments**
   While it is true that security cannot be tested into an application, application testing and assessments should still be a central component of an overall security strategy. Assessments—particularly automated tests—can find security problems not detected during code or implementation reviews, find security risks introduced by the operational environment, and act as a defense-in-depth mechanism by catching failures in design, specification, or implementation. Test and assessment functions are typically owned by a test analyst or by the QA organization but can span the entire life cycle.
3. **Capture security requirements**
   Ensure that security requirements have the same level of "citizenship" as all other "must haves." It's easy for application architects and project managers to focus on functionality when defining requirements, as they support the greater purpose of the application to deliver value to the organization. Security considerations can easily go by the wayside, so it is crucial that security requirements be an explicit part of any application development effort. Among the factors to be considered are:
   - an understanding of how applications will be used and how they might be misused or attacked.
   - the assets (data and services) that the application will access or provide and what level of protection is appropriate given the organization's appetite for risk, regulations to which it is subject, and the potential impact on its reputation should an application be exploited.
   - the architecture of the application and probable attack vectors.
   - potential compensating controls and their cost and effectiveness.
4. **Implement secure development practices**
   An essential application-security goal is to create and maintain reusable source code that strengthens basic security services—within an application and across an organization's applications. This goal is best achieved through implementing secure development practices into an organization's overall development process as early as possible in the SDLC.

   This CLASP Best Practice makes available an extensive set of process components. It provides well-defined, role-based activities that, for example, help guide project teams when applying security principles to design, integrating security analysis into the source management process, and implementing/elaborating resource policies and security technologies. It also provides extensive sample coding guidelines as well as artifacts like the 104 CLASP Problem Types that help a project team systematically avoid/resolve security vulnerabilities in source code.
5. **Build vulnerability remediation procedures**
   It is especially important in the context of application updates and enhancements to define which steps will be taken to identify, assess, prioritize, and remediate vulnerabilities. Building remediation procedures will speed response and minimize risk by defining roles, responsibilities, and processes to follow after identification of vulnerability. Remediation procedures are often fed by software assessments, both in house or third party, and help to control information when disclosure must occur.
6. **Define and monitor metrics**

A development project team cannot manage what it cannot measure. Unfortunately, implementing an effective metrics monitoring effort can be a difficult undertaking. Despite this, metrics are an essential element of an overall application security effort. They are crucial in assessing the current security posture of an organization, help focus attention on the most critical vulnerabilities, and reveal how well —or poorly—the organization's investments in improved security are performing.

7. **Publish operational security guidelines**
   Security does not end when an application is completed and deployed in a production environment. Making the most of existing network and operational security investments requires that those tasked with monitoring and managing the security of running systems are informed and educated; they need advice and guidance on the security requirements the application demands and how to best use the capabilities built into the application.

24 CLASP Activities

CLASP is designed to allow easy integration of its security-related activities into existing application development processes.

Each CLASP Activity is divided into discrete process components and linked to one or more specific project roles. In this way, CLASP provides guidance to project participants (e.g., project managers, security auditors, developers, architects, testers, and others) that is easily adaptable to their way of working. This results in incremental improvements to security that are easily achievable, repeatable, and measurable.

Note: The project role of security auditor is CLASP created. This role mainly examines the current state of a project and tries to ensure the security of the current state of the project in these project phases: requirements, design, and implementation.

Table 1 lists the 24 CLASP Activities and their related project roles and shows the recommended distribution of these activities across the CLASP Best Practices.

Table 1. CLASP activities and related project roles and best practices

CLASP Best Practices	CLASP Activities	Related Project Roles
1. Institute awareness programs	Institute security awareness program	• Project manager
2. Perform application assessments	Perform security analysis of system requirements and design (threat modeling)	• Security auditor
	Perform source-level security review	• Owner: security auditor • Key contributor: implementer, designer
	Identify, implement, and perform security tests	• Test analyst
	Verify security attributes of resources	• Tester
	Research and assess security posture of technology solutions	• Owner: designer • Key contributor: component vendor
3. Capture security requirements	Identify global security policy	• Requirements specifier
	Identify resources and trust boundaries	• Owner: architect • Key contributor: requirements specifier

	Identify user roles and resource capabilities	• Owner: architect • Key contributor: requirements specifier
	Specify operational environment	• Owner: requirements specifier • Key contributor: architect
	Detail misuse cases	• Owner: requirements specifier • Key contributor: stakeholder
	Identify attack surface	• Designer
	Document security-relevant requirements	• Owner: requirements specifier • Key contributor: architect
4. Implement secure development practices	Apply security principles to design	• Designer
	Annotate class designs with security properties	• Designer
	Implement and elaborate resource policies and security technologies	• Implementer
	Implement interface contracts	• Implementer
	Integrate security analysis into source management process	• Integrator
	Perform code signing	• Integrator
5. Build vulnerability remediation procedures	Manage security issue disclosure process	• Owner: project manager • Key contributor: designer
	Address reported security issues	• Owner: designer • Fault reporter
6. Define and monitor metrics	Monitor security metrics	• Project manager
7. Publish operational security guidelines	Specify database security configuration	• Database designer
	Build operational security guide	• Owner: integrator • Key contributor: designer, architect, implementer

CLASP Resources (including CLASP Keywords)

An integral part of the CLASP Process are resources that aid the planning, implementing, and performing CLASP Activities. Completing the sample coding guideline worksheets resource, for example, can help project managers understand which of the 104 CLASP Problem Types (see Vulnerability View above) could pose security risks for building a particular software system. This knowledge, in turn, can be used to help determine how CLASP Activities should be performed.

Several of the CLASP Resources are especially useful for projects that use tools to help automate CLASP process components.

These are the CLASP Resources, shown with examples.

- Basic principles of application security

    - insider threats as the weak link
    - assume the network is compromised
    - minimize attack surface
    - secure-by-default
    - defense-in-depth
    - principles for reducing exposure
    - insecure-bootstrapping principle
    - example of basic-principle violation: penetrate-and-patch model
- Descriptions of core security services and concepts
    - authorization (access control)
    - authentication
    - confidentiality
    - data integrity
    - availability
    - accountability
    - non-repudiation
    - Sample coding guidelines (worksheets)
    - build and test
    - network usage
    - authentication
    - input validation
    - file system
    - documentation
    - object-oriented programming
    - cryptography
    - UNIX-specific
    - Windows-specific
    - C, C++, Perl, Python, PHP
    - Java mobile code
    - Web applications
    - Generic mobile / untrusted code environments
- System assessment worksheets
    - development process and organization
    - system resources
    - network resource detail
    - file system usage detail
    - registry usage (Microsoft Windows environment)
- Sample road maps
    - legacy projects
    - new-start projects
- Aids for creating the process engineering plan
- Aids for forming the process engineering team
- Glossary of security terms

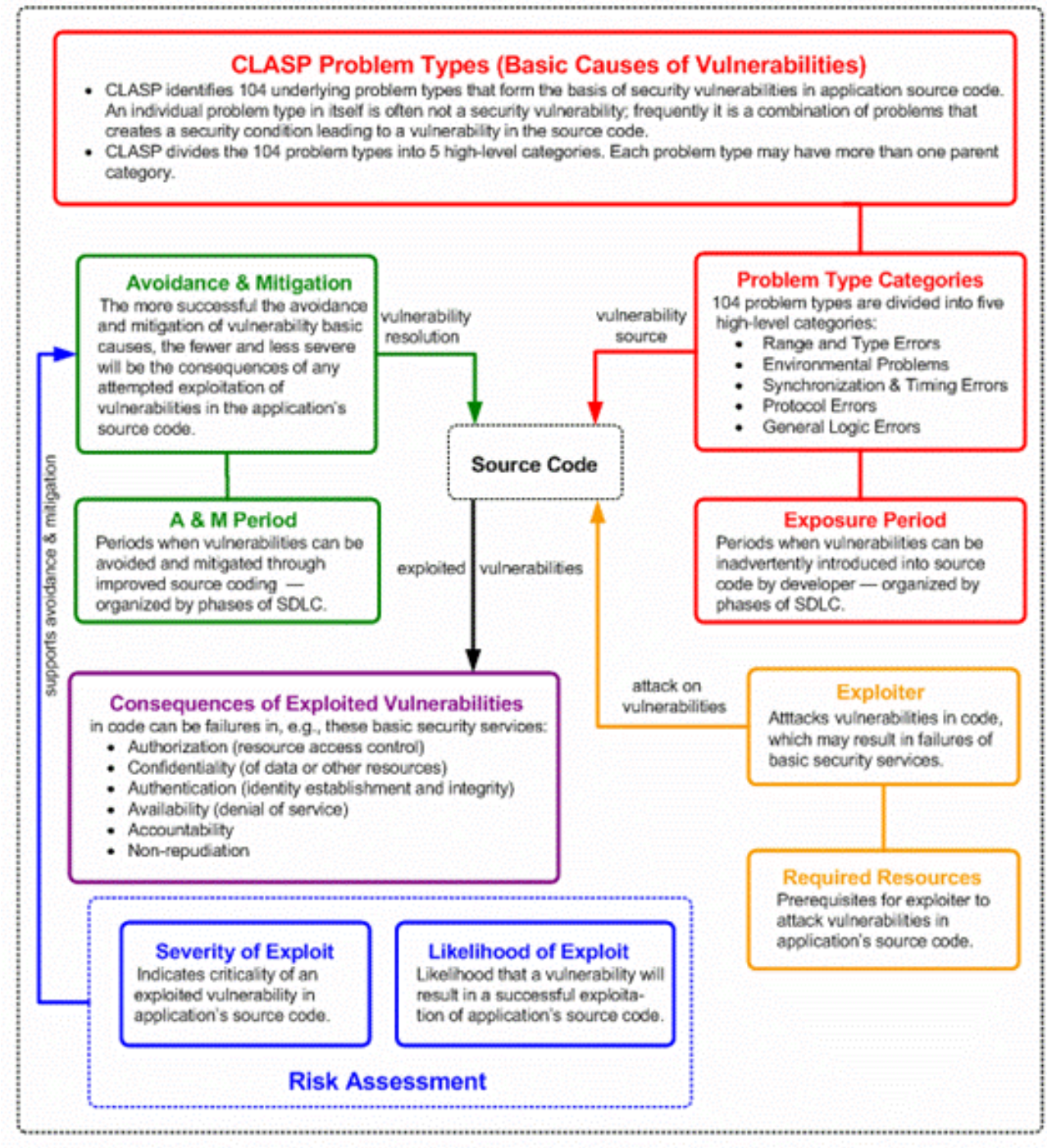
Taxonomy of CLASP

The CLASP Taxonomy is a high-level classification of the CLASP Process, which is divided into the following classes for better assessment and resolution of security vulnerabilities in source code:

- **Problem types** (i.e., basic causes) underlying security-related vulnerabilities.
- **Categories** into which the problem types are divided for diagnostic and resolution purposes. The five categories are as follows:
  - range and type errors
  - environmental problems
  - synchronization and timing errors
  - protocol errors
  - general logic errors
- **Exposure periods** (i.e., SDLC phases) in which vulnerabilities can be introduced into application source code.
- **Consequences** of exploited vulnerabilities for basic security services.
- **Resources** required by **exploiters** for attacks on security vulnerabilities.
- **Risk assessment** of exploitable/exploited vulnerabilities.
- **Avoidance and mitigation periods** (i.e., SDLC phases), in which preventative measures and countermeasures can be applied.

Figure 2 represents a high-level view of the CLASP Taxonomy.

Figure 2. High-level view of CLASP taxonomy



Further Information on CLASP

For more information, see the OWASP CLASP Project[5] article on the OWASP Web site.

Secure Software Copyright

5. http://www.owasp.org/index.php/Category:OWASP_CLASP_Project

1. mailto:info@securesoftware.com